Daniel Klingensmith
202580-ITCS-6162

Movie Recommendation System Report

Introduction

Estimates of video streaming revenue for 2024 range as high as ~700 billion dollars, and competition is high. Every company that can has their own streaming service in an attempt to slice a bit from that pie, and it can be tough to try and keep users in such a saturated market. As such, it's extremely important to these companies that customers see products that will keep their attention and their dollars. To that end, they all, understandably, work very hard on having a good movie/TV recommendation system, to provide the best customer experience they can (and presumably boost sales on platforms that have pay-per-view, like Amazone Prime).

This assignment involved creating several different basic recommender systems- specifically, a user-based, an item-based, and a Pixie-inspired random walk recommender. User based systems compare a target user's taste to the tastes of other users and then use those most similar to recommend movies. Item-based work by identifying items that are commonly associated together. A random walk system involves bouncing back and forth between sides of a partite graph of users and movies, seeing which items get touched the most, indicating which items might be considered closely related or relevant by users' interactions.

The Data

This assignment used data from the MovieLens 100k dataset. The data consists of three files: one with user-movie rating data, one with movies, and one with demographic data around the users. In all, the data contained 943 users, 1682 movies, and 10,000 ratings. Not all of this information was used; the main pieces of data that were used in this assignment were the movies ids and titles, and the user ids and movie ratings. The data came fairly well cleaned to begin with, with no missing or corrupted fields, so the only preprocessing performed was converting the timestamps on the ratings into a human readable format.

Methodology and Implementation

User-based collaborative filtering:

First, similarity between users was calculated using cosine similarity. A table of user-user similarity was created using this cosine similarity; missing values were set as 0, and then. When doing a recommendation, the top N most similar users are found. From these, scores are averaged (excluding 0s), and the highest average scores are returned to the user as recommendations.

Item-based filtering:

As with user based filtering, cosine similarity was used to estimate similarity between items, with missing values set as 0s. From here, it was just a matter of sorting movies with the highest similarity (excluding itself), and returning the amount of recommendations requested.

Pixie-inspired random walk:

Users and movies were assembled into a partite graph by iterating through the ratings data and for each user id - movie id pair, two entries were added to a dictionary, using the movie title and the user id as keys, and adding the other to the list of values stored under the key. Once the graph was assembled, the recommender would randomly walk (randomly jump to a connected node) a specified amount of steps from a starting source, either a movie or a user. This process was repeated a specified amount of times, and the amount of times a movie was accessed was counted. Movies were then sorted by the amount of times they were counted, and the highest counts were returned as recommendations.

Daniel Klingensmith
202580-ITCS-6162

Example outputs

User-based:

```
recommend_movies_for_user(10, num = 5, most_similar_user_count=50)
```

```
1,"Boys, Les (1997)"
2,Braindead (1992)
3,"Thin Blue Line, The (1988)"
4,In the Company of Men (1997)
5,"Misérables, Les (1995)"
```

Increasing the most_similar_user_count to a certain point made results more predictable (as there was a larger consensus pool), but expanding too far included low similarity results and begins to skew towards simply popular items.

The implementation used in this assignment used cosine similarity, which is not a great gauge for numerical results, and would likely have been better served by something like a Pearson Correlation Coefficient. It is also less effective for sparse user data; luckily, this dataset was curated and every user had a decent amount of recommendations to work with.

Item-based:

```
recommend_movies('GoldenEye (1995)', num=5)
```

```
1,Under Siege (1992)
2,Top Gun (1986)
3,True Lies (1994)
4,Batman (1989)
5,Stargate (1994)
```

The item based filtering shares user-based's limitations from cosine similarity. With movies, it's easier to get a decent spread of values for making the comparison, which can be useful, and could be supplemented in house for fresh releases and things that don't have much ratings data yet. It is a little less effective for people with unusual tastes, as it relies on common interactions with the movie, which can miss some nuances.

Random walk:

```
weighted_pixie_recommend("Jurassic Park (1993)", walk_length=10, num=5, walks_to_execute=100)
```

```
1,Star Wars (1977)
2,Independence Day (ID4) (1996)
3,Braveheart (1995)
4,Raiders of the Lost Ark (1981)
5,Forrest Gump (1994)
```

Small numbers of walks executed tend to suffer from a bit of randomness- sometimes wandering down unusual connection paths that represent irregular choices by users. Increasing the number of walks limits the amount of outliers, as they get averaged out, but also makes the recommendations more generic, as they lean towards popular items. This system also doesn't utilize rating data by itself, wasting that information, although a weighted random walk could probably be created to do so.

Conclusion

This assignment provides an excellent snapshot of the basics of core recommender system methodologies. While all of the algorithms were a bit simplified, they allow some of the pitfalls and limitations of each to be easily seen.

Since these were simplified, there are many things that could have been done to improve the efficacy of the algorithms. Better similarity calculations could have been done, incorporating ratings values and accounting for individual ratings bias in their rankings. The random walk system would have benefitted from taking advantage of ratings values with a weighted walk. A hybrid system between all three would almost certainly have been better than any one alone. Modern recommender systems are quite complicated, so the options for expanding this are almost limitless, from little changes like providing a baseline estimated value for nulls, to complex changes like incorporating demographic data to create a system that uses the extra metadata it provides in similarity calculations.

Each of the techniques utilized are viable in a range of places, though some are better suited than others. A simple random walk, for example, underutilizes ratings data, but is excellent for places that don't have that kind of information- something like TikTok, for example, which only have 'like' or nothing, or something like a dating app. Item based filters can be particularly useful for datasets with very sparse user ratings- book recommendations, for example, as the amount of books an average reader rates is likely to be much lower than other kinds of products. User-based filtering is especially good with the more user information you have available to calculate similarities; social media sites like Facebook or Twitter/X have a lot of info about their users that can make for deep similarities in recommending new content to follow.

As is often the case, though, a mix of techniques is almost always beneficial, and the datasets and context should be carefully considered before beginning implementations.

Citations:

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872